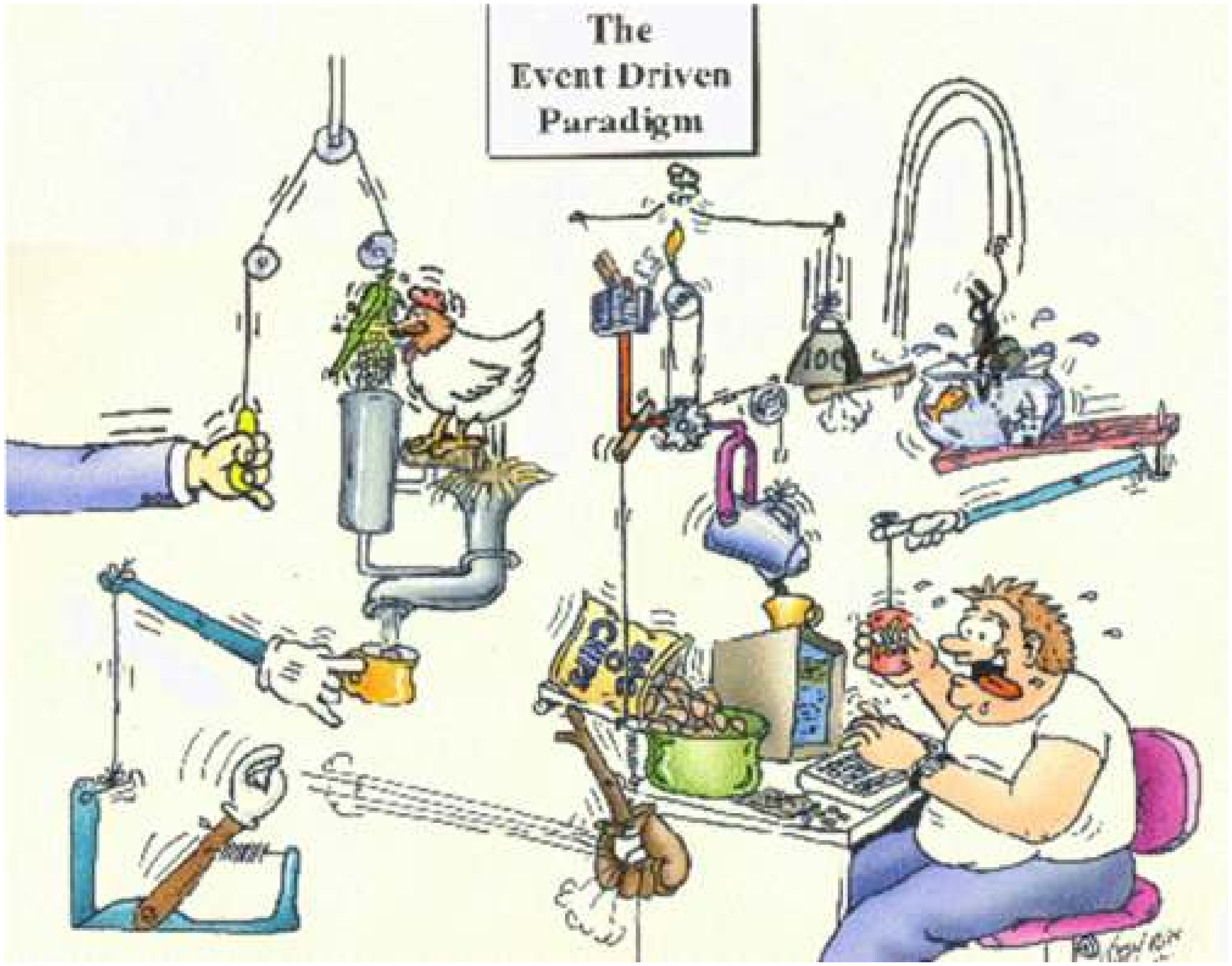
GUI and Web Programming

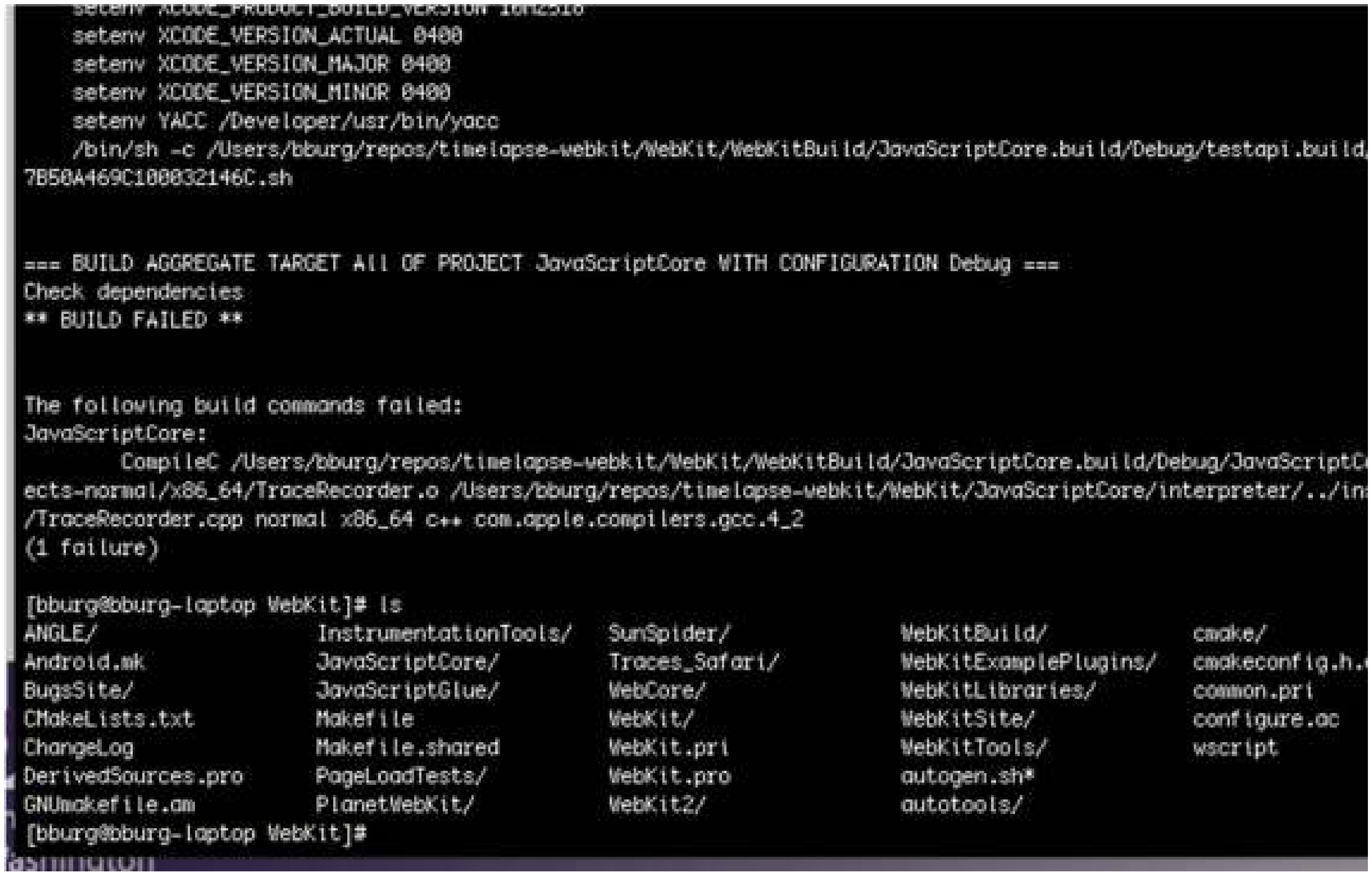
CSE 403

(based on a lecture by James Fogarty)

Event-based programming



Sequential Programs



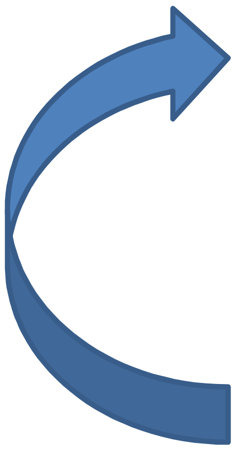
```
    setenv XCODE_PRODUCT_BUILD_VERSION 10M2518
    setenv XCODE_VERSION_ACTUAL 0400
    setenv XCODE_VERSION_MAJOR 0400
    setenv XCODE_VERSION_MINOR 0400
    setenv YACC /Developer/usr/bin/yacc
    /bin/sh -c /Users/bburg/repos/timelapse-webkit/WebKit/WebKitBuild/JavaScriptCore.build/Debug/testapi.build/
7B50A469C100032146C.sh


=== BUILD AGGREGATE TARGET All OF PROJECT JavaScriptCore WITH CONFIGURATION Debug ===
Check dependencies
** BUILD FAILED **


The following build commands failed:
JavaScriptCore:
        CompileC /Users/bburg/repos/timelapse-webkit/WebKit/WebKitBuild/JavaScriptCore.build/Debug/JavaScriptC
ects-normal/x86_64/TraceRecorder.o /Users/bburg/repos/timelapse-webkit/WebKit/JavaScriptCore/interpreter/../in
/TraceRecorder.cpp normal x86_64 c++ com.apple.compilers.gcc.4_2
(1 failure)

[bburg@bburg-laptop WebKit]# ls
ANGLE/                  InstrumentationTools/   SunSpider/              WebKitBuild/            cmake/
Android.mk              JavaScriptCore/         Traces_Safari/          WebKitExamplePlugins/   cmakeconfig.h.
BugsSite/               JavaScriptGlue/         WebCore/                WebKitLibraries/        common.pri
CMakeLists.txt          Makefile                WebKit/                 WebKitSite/             configure.ac
ChangeLog               Makefile.shared         WebKit.pri              WebKitTools/            wscript
DerivedSources.pro      PageLoadTests/          WebKit.pro              autogen.sh*
GNUmakefile.am          PlanetWebKit/           WebKit2/                autotools/
[bburg@bburg-laptop WebKit]#
```

Interacting with the user



- 1. Program takes control
- 2. Program does something
- 3. Program asks for **user** input
- 4. **User** provides input

The user as a file

- 1. Program takes control
- 2. Program does something
- 3. Program asks for **file** input
- 4. **File** provides input

The user is abstracted as a file (named `STDIN`)

Event-driven Programming

- User can provide input at <u>any</u> time

- User actions generate **events**

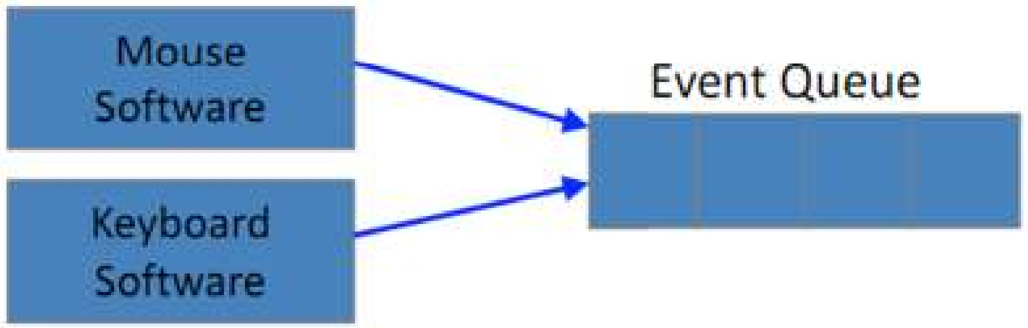
  mouse click/scroll/move, hover, key press, resize

  Event = type + button/position + target

Event Queues

All events go to an event queue

- provided by operating system
- Ensures events are handled in the order they occur
- hides specifics of input from apps



Event Queues

- How many event queues are there in modern desktop GUI environments?
- How can we tell without knowing the implementation details?
- What are the implications?

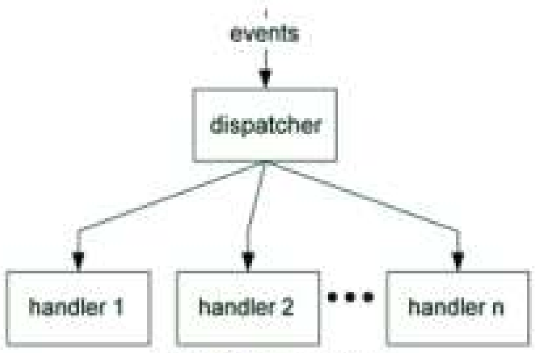
Interactive Software Loop

```
do {
        e = read_event();                   } input
        dispatch_event(e);
        if (damage_exists())                }
                update_display();           } output
} while (e.type != WM_QUIT);
```

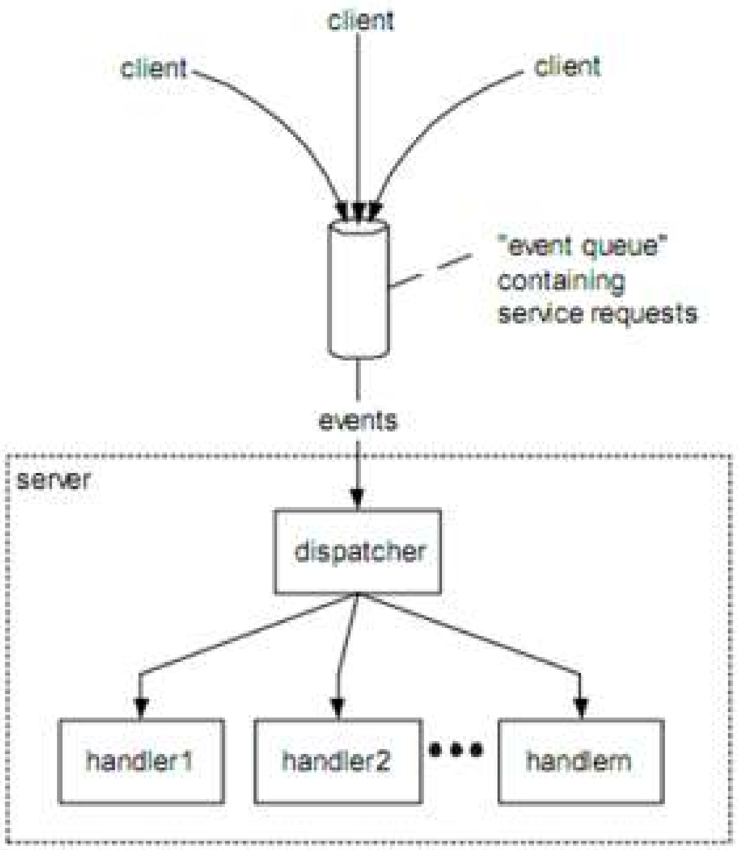
Nearly all GUI software has this somewhere

dispatch_event(e)

Keyboard,
mouse,
touchpad,
accelerometer



Handlers pattern



dispatch_event(e)

- Handlers (callbacks) are installed to register interest in some event type
- Dispatch notifies all handlers
- Also known as **Publish/Subscribe**, **Observer**

Model-View-Controller (MVC)

- (See CSE 510 slides; p22-31)

GUI Toolkits

- Reduce time necessary to create a UI
- Ready-made UI elements, events
- Windows Forms, GTK, QT, Cocoa, Swing, …
- Web pages! (more on this later)

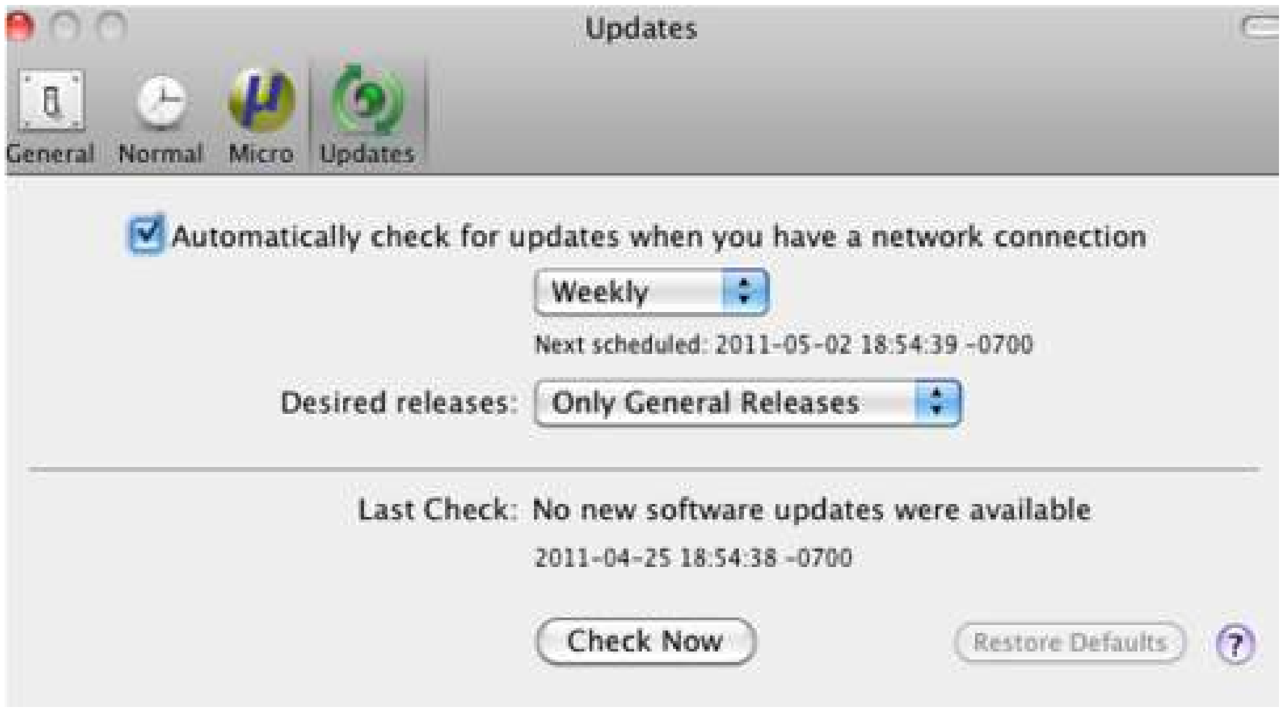
Typically, in a GUI Toolkit…

- Model backed by database, objects in memory, files

- View/Controller is merged

- Visual output based on tree of UI **elements** and their properties

Simple UI



[✓] Automatically check for updates when you have a network connection

Weekly

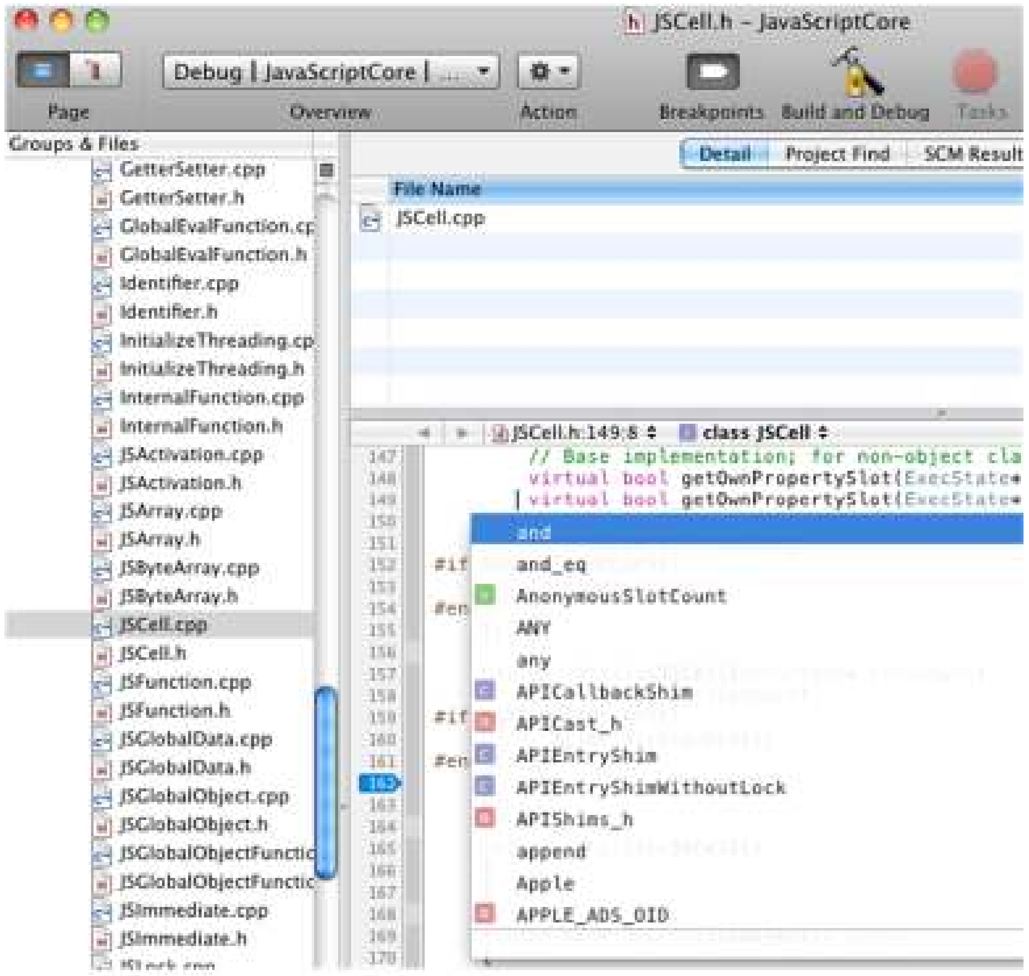
Next scheduled: 2011-05-02 18:54:39 -0700

Desired releases: Only General Releases

Last Check: No new software updates were available

2011-04-25 18:54:38 -0700

Less-simple UI



GetterSetter.cpp
GetterSetter.h
GlobalEvalFunction.cp
GlobalEvalFunction.h
Identifier.cpp
Identifier.h
InitializeThreading.cp
InitializeThreading.h
InternalFunction.cpp
InternalFunction.h
JSActivation.cpp
JSActivation.h
JSArray.cpp
JSArray.h
JSByteArray.cpp
JSByteArray.h
JSCell.cpp
JSCell.h
JSFunction.cpp
JSFunction.h
JSGlobalData.cpp
JSGlobalData.h
JSGlobalObject.cpp
JSGlobalObject.h
JSGlobalObjectFunctio
JSGlobalObjectFunctio
JSImmediate.cpp
JSImmediate.h

```
        // Base implementation; for non-object cla
        virtual bool getOwnPropertySlot(ExecState*
        virtual bool getOwnPropertySlot(ExecState*

#if
#en

#if
#en
```

```
and
and_eq
AnonymousSlotCount
ANY
any
APICallbackShim
APICast_h
APIEntryShim
APIEntryShimWithoutLock
APIShims_h
append
Apple
APPLE_ADS_OID
```

Painting UI elements

- Each UI element (component) is responsible for drawing itself and its children
- Typically event-based

```
void OnPaint(GraphicsContext g)
  //paint myself
  for (child in this.children) {
    child.paint(g);
  }
}
```

When to paint?

- The application does not decide!
- UI toolkits keep track of screen *damage*
- Toolkit will call paint() as necessary to fix “damage” to the bitmap
- Delegation of this greatly simplifies GUIs

How does damage happen?

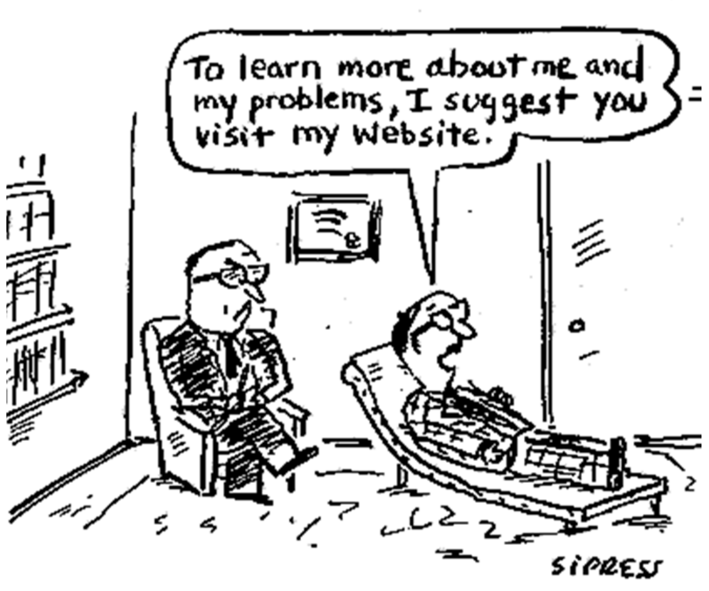
- By external (transparent) events
  - Mouse cursor, hidden window, overlap

- By dirtying part of the UI component tree
  - Component.invalidate() will damage the area occupied by the component, causing later repaint.

Routing user input/damage

- For mouse input, known as **hit testing**
  - Maps from an active pixel to a UI element
- For keyboard input, focus management
  - The element in “focus” receives keyboard events
- Delegation strategies vary per framework

Web (client) Programming


To learn more about me and my problems, I suggest you visit my website.
SIPRESS

HTML / CSS

- HTML = hypertext markup language
- A language for structuring and marking up documents in a semantic way
- Similar to LaTeX, PostScript

JavaScript

- Dynamically-typed scripting language
- Prototype-based object system
- Highly flexible and dynamic
- Transmitted only in source form

DOM / CSS

- DOM = document object model
- The abstract syntax tree of HTML
- Large API interacting with document tree
- CSS = cascading style sheets
  - Properties for DOM nodes based on pattern matching

HTML + JavaScript + DOM

- A GUI toolkit, with some catches
- DOM serves as model, view, and controller
- Event handlers written in JavaScript
- Visual output derived from DOM node props
  - No paint method!

Demo: Web page

- DOM as HTML AST
- Tree structure
- DOM node -> visual output
- CSS matches on DOM nodes
- Assembled from many pieces
- Damage => recompute styles, layout

Demo: Web application

- User input generates events
- Event handlers installed per DOM node
- Incremental repaint of “damaged” area
- Assembled from many pieces *dynamically*

AJAX?

- <u>**A**</u>synchronous <u>**J**</u>avaScript <u>**a**</u>nd <u>**X**</u>ML

- Supports loading JavaScript asynchronously
  - As opposed to forcing <script> load
  - Event/callback based

JavaScript Libraries?

- jQuery, Prototype, Scriptaculous
- Advantages:
  - Remove a lot of boilerplate DOM code
  - Alternate, browser-consistent API
- Disadvantages:
  - Difficult to debug a large library
  - Difficult to reuse code that uses one library

Pros and cons of web applications

- Pros:
  - Nothing to install, just need conformant browser
  - Easier to configure dynamically
  - Effortless "software update"
- Cons:
  - HTML/JS/DOM not intended for stateful apps
  - Usually requires internet connection
  - Less control over user experience

Web (server) Programming

- Can be implemented in any language
  - Popular: PHP, Ruby, Java, Python, Perl
- Web application does not care who speaks
  - Load balancing, proxies, firewalls
- All communication via HTTP requests
  - GET, POST, (PUT, DELETE)
  - Static resources *and* application requests

Web (server) Programming

- Each request is handled in isolation
  - But application itself must be highly concurrent, parallel to serve many users

- Step 1: Decode user request
- Step 2: Do something
- Step 3: Send response to user

Web (server) programming

- Architecture and protocols still fluid
- As always, many frameworks exist to ease application development
- Deserves its own lecture but..
  - Probably best to go read the web!

Bonus: Research

- Research at all points touching the web:
  - Debugging
  - Domain-specific languages
  - Application architecture
  - Testing
  - Performance
  - Security
  - HCI